

A Quantization-Friendly Hardware Implementation of the H-Swish Activation Function

Insu Jeong¹, Taejun Kim¹, Joowan Kang¹, Seung-Ho Ok², Dongsup Song³, and Byungin Moon^{1,4,*}

Abstract

In on-device environments, the computational efficiency of lightweight neural networks, such as MobileNet, is a critical factor for determining their performance. The h-swish activation function introduced in MobileNetV3 provides high accuracy. In FPGAs, multiplication operations are typically performed using DSP blocks. However, using DSP blocks for activation functions is inefficient in terms of hardware resource utilization, as these blocks are essential for convolution computations. To address this issue, we propose an approximation method and a hardware architecture for the h-swish activation function that eliminates multiplication operations, thereby avoiding the use of DSP blocks. The proposed architecture employs a piecewise linear (PWL) approximation to replace multiplication with hardware-friendly bit-shift and addition. Furthermore, we extend the conventional PWL approach to present a hardware implementation optimized for integer inputs. Experimental results confirm that the proposed h-swish hardware achieves a DSP-free implementation and, for integer inputs, reduces look-up table (LUT) usage by approximately 32% compared to conventional h-swish hardware implementation. When applied to MobileNetV3-Large, the proposed h-swish activation function incurs only a negligible loss in top-1 accuracy. Therefore, the proposed method provides an effective solution for efficient neural network inference on resource-constrained hardware, such as edge devices.

Keywords

FPGA, Quantization, Activation Function, Piecewise Linear Approximation

1. Introduction

Recently, the importance of on-device AI technology, which performs artificial intelligence computations directly on edge devices such as smartphones, autonomous vehicles, robots, and Internet of Things (IoT) devices, has been growing significantly. A key research challenge in this area is the design of lightweight neural networks, as these models must operate within the constrained computational resources and power constraints of edge devices.

MobileNet represents a typical lightweight neural network, utilizing depthwise separable convolution to reduce both the number of parameters and computational complexity while maintaining high accuracy. Therefore, MobileNet has been widely adopted across various applications, including real-time object

※ This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (<http://creativecommons.org/licenses/by-nc/3.0/>) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

Manuscript received January 13, 2026; accepted August 6, 2026.

*Corresponding Author: Byungin Moon (bihmoon@knu.ac.kr)

¹ Graduate School of Electronic and Electrical Engineering, Kyungpook National University, Daegu, Korea (insu8944@knu.ac.kr, ktj016669@knu.ac.kr, jwkang1231@knu.ac.kr, bihmoon@knu.ac.kr)

² Department of Robotics, Dong-eui University, Busan, Korea (osh@deu.ac.kr)

³ Department of Electronic Engineering, Hoseo University, ChungNam, Korea (dssong@hoseo.edu)

⁴ School of Electronics Engineering, Kyungpook National University, Daegu, Korea (bihmoon@knu.ac.kr)

detection and image classification, where fast inference speed is essential. MobileNet has evolved over successive versions with various architectural improvements, and advancements in activation functions have been a key contributor to its performance enhancement. The initial versions, MobileNetV1 [1] and MobileNetV2 [2], used ReLU and ReLU6 activation functions, which provide stable performance even on low-precision hardware.

Subsequently, MobileNetV3 [3] adopted the swish activation function to overcome the limitations of ReLU activation function (e.g., the Dying ReLU) and to improve accuracy. While swish effectively captures nonlinearity and enhances model performance, it suffers from a high computational cost due to its reliance on the sigmoid function. To address this, MobileNetV3 proposed the h-swish activation function, an approximation of swish that reduces complexity while maintaining accuracy. However, when implementing convolutional neural network (CNN) accelerators on field-programmable gate arrays (FPGAs), the h-swish function still poses challenges. Its formulation includes multiplication operations that, when mapped to FPGAs, require either embedded digital signal processing (DSP) blocks or a large number of look-up tables (LUTs). In CNN accelerators, DSP blocks are critical resources for executing convolution operations, which constitute the majority of the computational workload. Consequently, allocating DSP resources to an activation function, whose contribution to the overall computation is relatively small, results in inefficient hardware resource utilization.

In this paper, we propose a novel approximation method for the h-swish activation function and a hardware architecture that eliminates multiplications, thereby avoiding the use of DSP blocks. The proposed approach employs a piecewise linear (PWL) approximation by dividing the h-swish function into several segments and by replacing multiplication with hardware-friendly bit-shift and addition operations. In addition, we extend the conventional PWL approach and propose a hardware architecture optimized for integer inputs. The proposed design eliminates DSP usage for both fixed-point and integer inputs and achieves an approximate 32% reduction in LUT utilization for integer inputs. Furthermore, when the proposed h-swish activation function replaces the original activation in MobileNetV3, the top-1 accuracy loss is negligible.

The rest of this paper is organized as follows: Section 2 provides background and reviews previous work related to activation function approximation and quantization. Section 3 describes the proposed approximation method and hardware architecture. Section 4 analyzes the hardware resources of the proposed architecture and evaluates its impact on model performance, followed by the conclusion in Section 5.

2. Related Work

In this section, we review the existing approaches for the hardware implementation of activation functions and neural network quantization schemes. Section 2.1 examines various activation function approximation methods that have been developed for efficient hardware implementation. Section 2.2 discusses neural network quantization schemes for reducing computational latency and resource requirements on hardware platforms.

2.1 Activation Function Approximation Methods

Activation functions are fundamental components that determine model performance. However, those

involving exponential functions or division, such as sigmoid and swish, incur high computational costs and substantial hardware resource consumption when their mathematical formulations are implemented directly without approximation. Consequently, developing hardware-friendly approximations of activation functions to reduce computational cost and resource usage has become an active area of research.

PWL approximation is one of the most prominent approaches. This approach approximates a complex nonlinear function by dividing it into multiple linear segments, which allows complex multiplication and division operations to be replaced with hardware-friendly alternatives [4-6]. Several studies have applied the PWL approximation to efficiently implement various activation functions such as sigmoid, tanh, and Gaussian error linear unit (GELU), with optimizations reducing hardware costs through the use of approximate adders [7]. As these studies demonstrate, PWL approximation offers an effective balance between hardware cost and performance, providing high accuracy with a low implementation complexity. However, many PWL approaches primarily focus on reducing the arithmetic complexity of nonlinear functions without explicitly incorporating quantization parameters into the approximation formulation. In contrast, the proposed method considers affine quantization, enabling PWL approximation to be efficiently extended to integer only arithmetic.

In addition to PWL approximation, various other approximation methods have been proposed. One approach is to approximate activation functions using hardware-friendly operations to reduce implementation complexity. Choi et al. [8] proposed a hardware-friendly approximation of the swish activation function and demonstrated improved hardware efficiency. Another approach is the LUT-based method, which stores pre-computed function values in memory and retrieves outputs by indexing, providing minimal computation latency. However, this method faces the fundamental limitation of requiring exponentially increasing memory resources to achieve high accuracy [9]. Therefore, research is being conducted to optimize memory resources through enhanced methods such as exploiting the symmetry of activation functions [10]. Polynomial approximation, which uses methods like Taylor series expansion and similar methods, can achieve high accuracy. However, it has the drawback of requiring numerous multipliers, leading to significant hardware resource consumption [11].

2.2 Quantization

On edge devices, inference is typically accelerated by applying a quantization scheme that converts floating-point weights and activations into integers. This method maps floating-point values to integers at uniform intervals, reducing the model storage and improving inference speed through the use of hardware optimized for integer operations.

Uniform quantization is fundamentally based on an affine transformation that maps floating-point real numbers to integer values. This transformation utilizes two parameters known as scale and zero-point. The scale determines the step size of the quantization, which defines the ratio for compressing the range of floating-point values into the integer range. The zero-point specifies the integer value to which the floating-point value of 0 is mapped, ensuring that the actual value zero can be represented without quantization error. This is crucial for maintaining accuracy in operations where the value 0 is significant, such as in the zero padding of convolution operations.

Jacob et al. [12] proposed an 8-bit integer quantization using the constants S (for “scale”) and Z (for “zero-point”). They introduced the method that allows for integer-arithmetic-only inference for neural networks. This is made possible by the quantization formula, as defined in Eqs. (1) and (2):

$$q = \text{round}\left(\frac{r}{S} + Z\right) \quad (1)$$

$$r = S(q - Z) \quad (2)$$

Eq. (1) defines the quantization process, converting a real value r into a quantized value q . First, the real value is divided by the constant S to adjust its range to the integer space, and then the constant Z is added to correct the offset. Conversely, Eq. (2) is the de-quantization formula that reconstructs an approximation of the original real value from the quantized value.

This bidirectional mapping formula allows a trained floating-point model to be converted into an integer model with minimal accuracy loss. This approach enables most of the complex floating-point operations during inference to be replaced with integer arithmetic operations, leading to its widespread adoption as a standard quantization scheme in deep learning frameworks.

In particular, to efficiently handle complex nonlinear activation functions in a quantized environment, Kim et al. [13] approximated the exponential function and GELU using integer bit-shifts and multiplication operations. However, these studies did not address the issue of activation functions with multiplication operations occupying hardware resources. Therefore, this paper, while following the fundamental principles of uniform quantization, focuses on optimizing h-swish for hardware resource constraints.

3. The Proposed Method

In this section, we propose a PWL approximation for the h-swish function and present its corresponding FPGA-based hardware architecture that utilizes only LUTs.

3.1 Piecewise Linear Approximation of H-Swish Function

The h-swish function is defined as the input x multiplied by the h-sigmoid function, as expressed in Eq. (3). This formulation inherently requires a multiplication operation.

$$h - \text{swish} = \begin{cases} 0, & x < -3 \\ x \cdot \frac{(x + 3)}{6}, & -3 \leq x < 3 \\ x, & x \geq 3 \end{cases} \quad (3)$$

The proposed method applies PWL approximation to the h-swish function within the interval $(-3, 3)$. This interval is partitioned into eight segments of width 0.75. Each segment is approximated by the linear function connecting its start and end points. The resulting PWL formulation eliminates the division by 6 and yields coefficients that can be exactly represented in fixed-point format without additional approximation error, enabling a multiplier-free implementation using only bit-shift and addition operations. The resulting approximated function is expressed in Eq. (4).

As illustrated in Fig. 1, the proposed approximation provides an accurate representation of the h-swish function across all intervals, demonstrating excellent agreement between the theoretical curve and the approximated segments. Moreover, the elimination of the division by 6 allows all constants in Eq. (4) to be expressed in a fixed-point format with 1 integer bit and 4 fractional bits, as shown in Eq. (5):

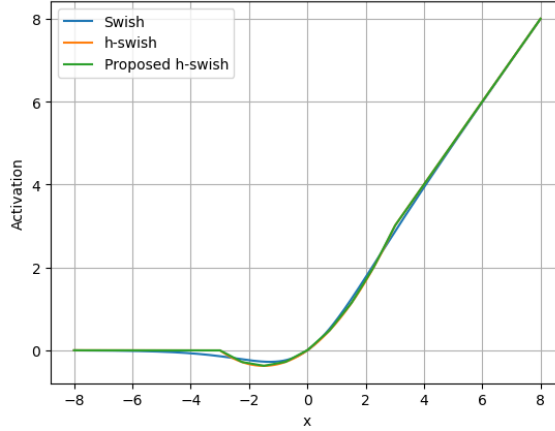


Fig. 1. Comparison of activation functions.

$$\text{proposed } h - \text{swish} = \begin{cases} 0, & x < -3 \\ -0.375x - 1.125, & -3 \leq x < -2.25 \\ -0.125x - 0.5625, & -2.25 \leq x < -1.5 \\ 0.125x - 0.1875, & -1.5 \leq x < -0.75 \\ 0.375x, & -0.75 \leq x < 0 \\ 0.625x, & 0 \leq x < 0.75 \\ 0.875x - 0.1875, & 0.75 \leq x < 1.5 \\ 1.125x - 0.5625, & 1.5 \leq x < 2.25 \\ 1.375x - 1.125, & 2.25 \leq x < 3 \\ x, & x \geq 3 \end{cases} \quad (4)$$

$$\text{proposed } h - \text{swish} = \begin{cases} 0, & x < -3 \\ -(0.011)_2x - (1.0010)_2, & -3 \leq x < -2.25 \\ -(0.001)_2x - (0.1001)_2, & -2.25 \leq x < -1.5 \\ (0.001)_2x - (0.0011)_2, & -1.5 \leq x < -0.75 \\ (0.011)_2x, & -0.75 \leq x < 0 \\ (0.101)_2x, & 0 \leq x < 0.75 \\ (0.111)_2x - (0.0011)_2, & 0.75 \leq x < 1.5 \\ (1.001)_2x - (0.1001)_2, & 1.5 \leq x < 2.25 \\ (1.011)_2x - (1.0010)_2, & 2.25 \leq x < 3 \\ x, & x \geq 3 \end{cases} \quad (5)$$

Since multiplication operations require only 1 integer bit and 3 fractional bits, they can be efficiently computed using up to 3 right shifts.

3.2 Hardware Implementation

3.2.1 Hardware implementation with fixed-point arithmetic

We implemented the proposed hardware architecture for the approximated h-swish function. For comparison, we also implemented a reference architecture of the conventional h-swish function without PWL approximation on the FPGA.

Fig. 2(a) shows the hardware architecture of the h-swish function. Within the input range $(-3,3)$, the h-swish can be implemented using Eq. (3). The input value x is added to the constant 3, while simultaneously the input value x is multiplied by $\frac{1}{6}$ to obtain $\frac{1}{6}x$. Subsequently, the two results are

multiplied together, and this result is used as input to the multiplexer. The final output is selected from three candidate values $(0, \frac{x \cdot (x+3)}{6}, x)$ depending on the input range.

Fig. 2(b) illustrates the hardware architecture of the proposed h-swish function. As presented in Eq. (5), divisions by 2, 4, and 8 are implemented using right shift operations by 1, 2, and 3 bits respectively, and the final result is produced by summing these shifted values with a constant selected depending on the input range. The Shift Selector takes the input x and its inverted version as inputs. The Shift Selector outputs five values based on the input range, as defined in Eq. (5). These outputs are summed by a carry save adder (CSA) to minimize propagation delay.

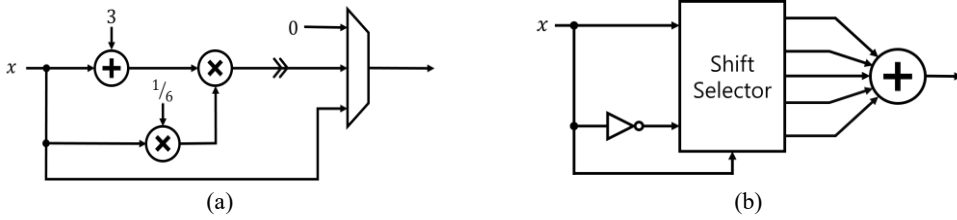


Fig. 2. Hardware architecture of (a) the h-swish and (b) the proposed h-swish.

The Shift Selector is composed of multiplexers, and its structure is detailed in Fig. 3. The output of each multiplexer is determined by the input range. The first output selects 0 for the input range $(-\infty, 0)$, $x \gg 1$ for $[0, 1.5)$, and x for $[1.5, \infty)$. The second output selects 0 for the intervals $(-\infty, -3)$, $[-2.25, -0.75)$, $[0, 0.75)$, $[1.5, 2.25)$, and $[3, \infty)$. For all other intervals, it selects either $x \gg 2$ or $\tilde{x} \gg 2$. For the third output, 0 is selected for $(-\infty, -3)$ and $[3, \infty)$, while $\tilde{x} \gg 3$ is selected for $[-3, -1.5)$ and $x \gg 3$ is selected for $[-1.5, 3)$. The fourth output generates a value to add 1 at a specific position to generate $-x$ as a two's complement from the inverted input. Finally, the fifth output selects among the constant values 0, 0.1875, 0.5625, and 1.125 depending on the input range. These five values are then passed to the subsequent adder stage.

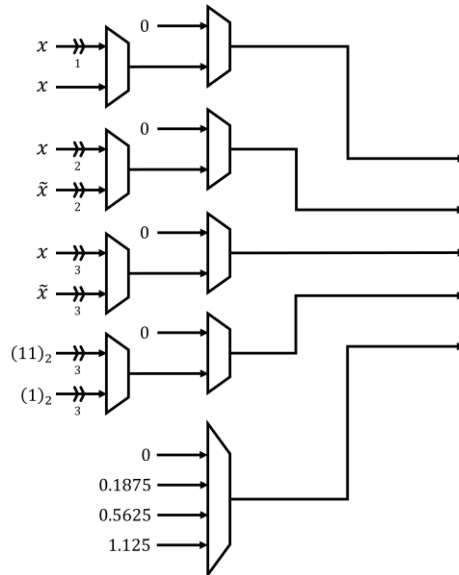


Fig. 3. Hardware architecture of Shift Selector.

3.2.2 Hardware Implementation with integer only arithmetic

The proposed h-swish function is implemented as a linear function in each segment. Therefore, for integer only arithmetic, the proposed h-swish can be expressed by Eq. (6) for each segment. In this equation, R_i represents each segment for the PWL approximation, as defined in Eq. (7). The $I_A(s_x(x_{int} - z_x) \in R_i)$ term is an indicator function that evaluates to 1 if the input $s_x(x_{int} - z_x)$ is within the specific interval R_i , and 0 otherwise, as shown in Eq. (7). This function acts as a switch that selectively activates only the term corresponding to the interval containing the current input value. The coefficients a_i and b_i are predefined constants that define the linear function for each segment R_i .

$$s_y(y_{int} - z_y) = \sum_{i=0}^9 I_A(s_x(x_{int} - z_x) \in R_i) \cdot \{a_i \cdot s_x(x_{int} - z_x) + b_i\} \quad (6)$$

$$I_A(x \in R_i) = \begin{cases} 1, & x \in R_i \\ 0, & x \notin R_i \end{cases} \quad R_i \in \{(-\infty, -3), [-3, -2.25), \dots, [2.25, 3), [3, \infty)\} \quad (7)$$

$$s_y(y_{int} - z_y) \approx \sum_{i=0}^9 I_A(s_x(x_{int} - z_x) \in R_i) \cdot \left[s_x \cdot \left\{ a_i \cdot (x_{int} - z_x) + \text{round}\left(\frac{b_i}{s_x}\right) \right\} \right] \quad (8)$$

$$y_{int} \approx \sum_{i=0}^9 I_A(s_x(x_{int} - z_x) \in R_i) \cdot \left[\frac{s_x}{s_y} \cdot \left\{ a_i \cdot (x_{int} - z_x) + \text{round}\left(\frac{b_i}{s_x}\right) \right\} \right] + z_y \quad (9)$$

The term $a_i \cdot s_x(x_{int} - z_x) + b_i$ in Eq. (6) combines integer and real number operations, resulting in computational inefficiency. To address this, we restructure the equation by factoring out the scale factor s_x , allowing internal operations to be performed using integer arithmetic and improving computational efficiency. Specifically, we mathematically transform the b_i term into the equivalent form $s_x \cdot \left(\frac{b_i}{s_x}\right)$, then factor out s_x yielding $s_x \cdot (a_i(x_{int} - z_x) + \frac{b_i}{s_x})$. At this stage, the term $\frac{b_i}{s_x}$ still involves a floating-point operation. Therefore, we apply the $\text{round}()$ function to approximate it to the nearest integer. Since b_i and s_x are known constants, the term $\text{round}(\frac{b_i}{s_x})$ can be pre-computed and stored in memory as a constant.

We rearrange Eq. (8) as follows to derive the expression for y_{int} , the final output to be computed in hardware. First, both sides of Eq. (8) are divided by the output scale factor s_y . In this process, the s_x on the right-hand side is divided by s_y , defining a new scale factor $\frac{s_x}{s_y}$. Since both s_x and s_y are known constants, this new factor can be pre-computed and stored in memory, thus avoiding the need for a division operation. Next, z_y is moved from the left to the right side of the equation, yielding the final expression for y_{int} as shown in Eq. (9). By incorporating quantization parameters into the PWL formulation in this manner, the proposed method can be efficiently extended to integer-only arithmetic.

Fig. 4 depicts the hardware architecture of the proposed integer arithmetic h-swish, which is designed based on Eq. (9). The computation first subtracts z_x from the integer input x_{int} to obtain $x_{int} - z_x$. The subsequent step is identical to the architecture of the proposed h-swish shown in Fig. 2(b), where the PWL operation is performed by a Shift Selector and the CSA. Finally, the summed result is forwarded to the Quantization Module. In this module, the final integer output y_{int} is generated by multiplying the result by the pre-computed $\frac{s_x}{s_y}$ and then adding the output zero-point z_y , according to Eq. (9).

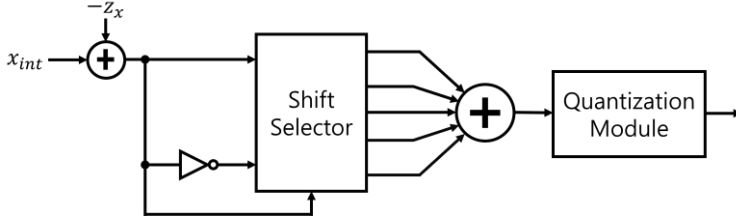


Fig. 4. Hardware architecture of the proposed integer arithmetic h-swish.

Fig. 5 illustrates the hardware architecture of the quantization module. This architecture is designed based on the constant coefficient multiplication (CCM) structure proposed by Li et al. [14]. To optimize hardware resources, this module approximates the floating-point number $\frac{s_x}{s_y}$ into the form $m \cdot 2^{-n}$. Here, m is a fixed-point number constant, and the 2^{-n} is replaced by an n -bit right-shift operation. The constant m and n are pre-computed. Therefore, no additional computational overhead occurs during inference.

For comparison, we also implemented a hardware architecture for the h-swish function operating entirely with integer arithmetic. As shown in Eq. (3), the h-swish function includes a quadratic term for inputs within the interval $[-3, 3]$, which was implemented using the method proposed by Kim et al. [13]. The integer arithmetic h-swish was constructed by connecting a quantization module to its output stage. This quantization module was implemented using the same method as that of the proposed h-swish.

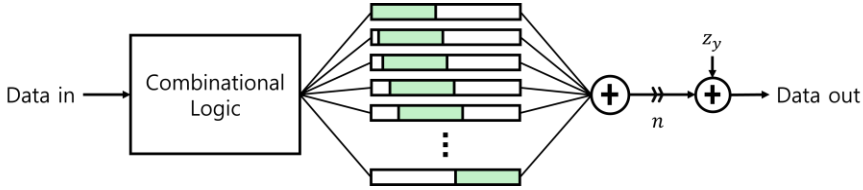


Fig. 5. Hardware architecture of the Quantization Module.

4. Evaluation

In this section, we evaluate the proposed approximation in terms of MSE (mean square error), model accuracy, hardware resource utilization, and latency. Table 1 presents the results of the MSE analysis for the activation functions, computed using 4,096 samples over an input range of $[-4.0, 4.0]$. The MSE between the proposed h-swish and the conventional h-swish function is 1.089×10^{-4} , indicating the high approximation accuracy of the proposed method. Furthermore, the MSE of the proposed h-swish with respect to the swish (3.322×10^{-3}) is smaller than that of the conventional h-swish (3.683×10^{-3}). Further hardware simplification can reduce resource usage at the cost of increased approximation error.

Table 1. MSE comparison of activation functions

Original function	Approximation function	MSE
h-swish	Proposed h-swish	1.089×10^{-4}
Swish	h-swish	3.683×10^{-3}
Swish	Proposed h-swish	3.322×10^{-3}

Table 2 compares the top-1 accuracy of MobileNetV3-Large using the conventional h-swish and the proposed h-swish at FP32 and INT8 precisions. MobileNetV3-Large was trained on ImageNet [15] for 600 epochs using stochastic gradient descent (SGD) with a momentum of 0.9, a weight decay of 1×10^{-5} , and an initial learning rate of 0.05. A linear warm-up was applied for the first 5 epochs, followed by cosine annealing. Cross-entropy loss with label smoothing was used, and the batch size was set to 128. For accuracy evaluation, the arithmetic operations of the proposed hardware architecture were faithfully reproduced in software to ensure the same computational behavior as the actual hardware implementation. The experimental results demonstrate that the proposed h-swish function achieves accuracy comparable to that of the conventional h-swish in both precision settings. In the FP32 model, the proposed h-swish function recorded an accuracy of 74.92%, confirming negligible degradation compared to 75.07% of the conventional h-swish. In the INT8 model, the proposed h-swish achieved an accuracy of 73.76%, slightly higher than 73.16% of the conventional h-swish. This difference is considered to result from variations introduced during the quantization process rather than a significant performance difference between the two activation functions.

Table 2. Comparison of the top-1 accuracy on MobileNetV3-Large

Data type	Activation function	Top-1 accuracy (%)
FP32	h-swish	75.07
	Proposed h-swish	74.92
INT8	h-swish	73.16
	Proposed h-swish	73.76

Table 3. Comparison of hardware resource for fixed-point design

Activation function	Bit width	Resource		
		LUT	FF	DSP
h-swish	8-bit	44	8	1
	12-bit	23	12	2
	16-bit	29	16	2
h-swish without DSP	8-bit	197	8	0
	12-bit	411	12	0
	16-bit	554	16	0
Choi et al. [8]	8-bit	8	8	1
	12-bit	10	12	1
	16-bit	12	16	1
Choi et al. [8] without DSP	8-bit	73	8	0
	12-bit	162	12	0
	16-bit	261	16	0
Proposed h-swish	8-bit	72	8	0
	12-bit	173	12	0
	16-bit	226	16	0

Table 4. Comparison of hardware resource for integer design

Activation Function	Data Type	Resource		
		LUT	FF	DSP
h-swish	INT8	373	8	1
Choi et al. [8]	INT8	289	8	1
Proposed h-swish	INT8	253	8	0

Table 3 summarizes the synthesis results for the fixed-point implementations, while Table 4 presents the results for the integer arithmetic implementations. All three designs were implemented on the xc7z020clg484-1 FPGA. In general, LUT utilization increases with bit width. However, the 8-bit conventional h-swish hardware requires more LUTs than the 12 and 16-bit implementations because the synthesis tool implements small-bit multipliers using LUTs instead of DSP blocks. The proposed h-swish hardware achieves a DSP-free implementation. For the DSP-free fixed-point implementations, the proposed method shows comparable resource utilization to Choi et al. [8] without DSP blocks, but requires fewer resources in the integer arithmetic implementation. These results demonstrate that the proposed method achieves low hardware resource utilization through its computation optimized for quantization, making it more efficient for integer arithmetic.

Table 5 compares the hardware latency of all three designs for fixed-point and integer arithmetic implementations. For the fixed-point implementations at 8, 12, and 16-bit precisions, the proposed h-swish hardware exhibits lower latency due to its simplified shift-and-add computation without multiplier operations. In the integer arithmetic implementation, the proposed h-swish hardware also achieves lower latency than the other methods. These results demonstrate that the proposed method provides efficient latency through its simplified computation and quantization-friendly design.

Table 5. Comparison of hardware latency

Activation function	Data type	Bit width	Latency (ns)
h-swish	Fixed-point	8-bit	6.57
		12-bit	8.90
		16-bit	9.26
Choi et al. [8]	Integer	8-bit	26.14
	Fixed-point	8-bit	5.68
		12-bit	7.08
16-bit		7.15	
Proposed h-swish	Integer	8-bit	25.11
	Fixed-point	8-bit	5.28
		12-bit	6.23
16-bit		7.05	
Proposed h-swish	Integer	8-bit	17.58

5. Conclusion

In this paper, we addressed the inefficiency problem of the h-swish activation function consuming DSP blocks, which are critical resources in FPGA-based CNN accelerators. To solve this problem, we proposed a hardware architecture that replaces the multiplication in the h-swish function with bit-shift and addition using PWL approximation. Furthermore, the proposed architecture is designed with a computational flow optimized for the INT8 quantization environment. FPGA synthesis results show that the proposed architecture requires no DSP blocks and, under INT8 quantization, also reduces LUT utilization. Moreover, when applied to the MobileNetV3-Large model, the proposed activation function maintains comparable Top-1 accuracy to the conventional h-swish in both FP32 and INT8 configurations. Consequently, the hardware approximation method proposed in this study offers a significant contribution to the design of efficient deep learning accelerators, enabling high-performance neural network inference on resource-constrained hardware, such as edge devices and embedded systems.

Conflict of Interest

The authors declare that they have no competing interests.

Funding

None.

References

- [1] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017 [Online]. Available: <https://doi.org/10.48550/arXiv.1704.04861>.
- [2] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proceedings of 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT, USA, 2018, pp. 4510–4520. <https://doi.org/10.1109/CVPR.2018.00474>
- [3] A. Howard, M. Sandler, G. Chu, L. C. Chen, B. Chen, M. Tan, et al., "Searching for MobileNetV3," in *Proceedings of 2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Seoul, Korea (South), 2019, pp. 1314–1324. <https://doi.org/10.1109/ICCV.2019.00140>
- [4] L. Wei, J. Cai, V. Nguyen, J. Chu, and K. Wen, "P-SFA: probability based sigmoid function approximation for low-complexity hardware implementation," *Microprocessors and Microsystems*, vol. 76, article no. 103105, 2020. <https://doi.org/10.1016/j.micpro.2020.103105>
- [5] X. Liao, T. Zhou, L. Zhang, X. Hu, and Y. Peng, "A method for calculating the derivative of activation functions based on piecewise linear approximation," *Electronics*, vol. 12, no. 2, article no. 267, 2023. <https://doi.org/10.3390/electronics12020267>
- [6] M. E. Sadeghi, A. Fayyazi, S. Azizi, and M. Pedram, "PEANO-ViT: power-efficient approximations of nonlinearities in Vision Transformers," in *Proceedings of the 29th ACM/IEEE International Symposium on Low Power Electronics and Design*, Newport Beach, CA, USA, 2024, pp. 1–6. <https://doi.org/10.1145/3665314.3670843>
- [7] C. Wu, Y. Yuan, W. Shi, C. Huang, X. Liu, and G. Li, "An PWL based fast efficient approximate sigmoid circuit design for activation function," in *Proceedings of 2024 4th Asia Conference on Information Engineering (ACIE)*, Singapore, 2024, pp. 111–116. <https://doi.org/10.1109/ACIE61839.2024.00025>
- [8] K. Choi, S. Kim, J. Kim, and I. C. Park, "Hardware-friendly approximation for swish activation and its implementation," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 71, no. 10, pp. 4516–4520, 2024. <https://doi.org/10.1109/TCSII.2024.3394806>
- [9] T. Sledevic and D. Navakauskas, "FPGA implementation of range addressable activation function for lattice-ladder neuron," *Elektronika ir Elektrotechnika*, vol. 22, no. 2, pp. 92–95, 2016. <https://doi.org/10.5755/j01.eie.22.2.14598>
- [10] R. Pogiri, S. Ari, and K. K. Mahapatra, "Design and FPGA implementation of the LUT based sigmoid function for DNN applications," in *Proceedings of 2022 IEEE International Symposium on Smart Electronic Systems (iSES)*, Warangal, India, 2022, pp. 410–413. <https://doi.org/10.1109/iSES54909.2022.00090>
- [11] T. Hoang and N. L. H. Trong, "Hardware implementation and optimization of the sigmoid activation function in CNNs using Taylor series approximation," in *Proceedings of 2024 International Conference on Advanced*

- Technologies for Communications (ATC)*, Ho Chi Minh City, Vietnam, 2024, pp. 625-630. <https://doi.org/10.1109/ATC63255.2024.10908160>
- [12] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT, USA, 2018, pp. 2704-2713. <https://doi.org/10.1109/CVPR.2018.00286>
- [13] S. Kim, A. Gholami, Z. Yao, M. W. Mahoney, and K. Keutzer, "I-BERT: integer-only BERT quantization," 2021 [Online]. Available: <https://doi.org/10.48550/arXiv.2101.01321>.
- [14] Z. Li, Z. Zhang, J. Hu, Q. Meng, X. Shi, and J. Luo, "A high-performance pixel-level fully pipelined hardware accelerator for neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 5, pp. 7970-7983, 2025. <https://doi.org/10.1109/TNNLS.2024.3423664>
- [15] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, et al., "ImageNet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211-252, 2015. <https://doi.org/10.1007/s11263-015-0816-y>



Insu Jeong <https://orcid.org/0009-0002-0134-2363>

He received the B.S. degree in Electronic Engineering from Kyungpook National University, Daegu, Korea, in 2023, where he is currently working toward the M.S. degree in Graduate School of Electronic and Electrical Engineering. His main interests are SoC, VLSI design, and computer vision.



Taejun Kim <https://orcid.org/0009-0003-2500-0277>

He received the B.S. degree in Electronic Engineering from Dong-Eui University, Busan, Korea, in 2024. He is currently working toward the M.S. degree in the Graduate School of Electronic and Electrical Engineering at Kyungpook National University, Daegu, Korea. His main interests are SoC, computer vision, and embedded systems.



Joowan Kang <https://orcid.org/0009-0008-2787-3588>

He received the B.S. degree in Electronic Engineering from Kumoh National Institute of Technology, Gumi, Korea, in 2024. Since 2024, he has been working toward the M.S. degree in the Graduate School of Electronic and Electrical Engineering, Kyungpook National University, Daegu, Korea. His research interests include SoC, VLSI design, and computer vision.



Seung-Ho Ok <https://orcid.org/0000-0002-9036-0872>

He received the B.S. degree in Mechatronics Engineering from Dong-eui University, Busan, Korea, in 2006, and the M.S. and Ph.D. degrees in Electrical Engineering and Computer Science from Kyungpook National University, Daegu, Korea, in 2008 and 2014 respectively. From 2011 to 2013, he was a Visiting Scholar at the GTCAD (Georgia Tech Computer-Aided Design) Laboratory at the Georgia Institute of Technology, Atlanta, GA, USA. He then worked as a Senior Engineer at Samsung Electronics from 2014 to 2017. Since September 2017, he has been a professor in the Department of Robotics at Dong-eui University. His research interests include on-device AI, physical AI, and system-on-chip design.



Dongsup Song <https://orcid.org/0009-0005-2426-4865>

He received the B.S. degree from the Department of Electrical Engineering, Konkuk University, Seoul, Korea, in 2000, the M.S. and Ph.D. degrees in Electrical and Electronics Engineering, Yonsei University, Seoul, Korea, in 2002 and 2007, respectively. He was a Principal DFT engineer with Samsung Electronics, Korea, from 2007 to 2021 and a Senior Staff Engineer in the EDA Group of Synopsys Korea from 2021 to 2024. In 2024, he joined the Faculty of Electronic Engineering, Hoseo University, ChungNam, Korea, where he is currently a professor. His research interests include AI driven DFT, 3D-IC DFT, DFT for automotive, and high quality DFT.



Byungin Moon <https://orcid.org/0000-0002-8102-4818>

He received the B.S. and M.S. degree in Electrical Engineering, and the Ph.D. degree in Electrical & Electronic Engineering from Yonsei University, Seoul, Korea, in 1995, 1997, and 2002, respectively. He spent two years as a senior researcher in Hynix Semiconductor Inc., and also worked as a research professor in Yonsei University for one year. Since 2005, he has been a professor in the School of Electronics Engineering, Kyungpook National University, Daegu, Korea. His main interests are SoC, computer architecture, and computer vision.